گزارش پنجم، دستور آزمایش چهارم

شهزاد ممیز ۸۱۰۱۰۰۲۷۲

محمد امانلو ۸۱۰۱۰۰۰۸۴

Part1. VNC Servers

Raspberry Pi یک کامپیوتر کوچک و قابل حمل است که می توان از آن برای طیف وسیعی از کاربردها استفاده کرد. یکی از کاربردهای Raspberry Pi، استفاده از آن به عنوان یک سرور VNC است. VNC یک پروتکل شبکه است که به شما امکان می دهد رابط کاربری گرافیکی (GUIیک رایانه را از راه دور کنترل کنید.

در این گزارش، مراحل راه اندازی VNC Server روی Raspberry Pi توضیح داده شده است.

مراحل راه اندازی:

۱. **نصب VNC Server روی Raspberry Pi**

برای نصب VNC Server روی Raspberry Pi، ابتدا باید یک اتصال SSH به Raspberry Pi برقرار کنید.

برای این کار، می توانید از یک ترمینال یا یک برنامه SSH مانند PuTTY استفاده کنید.

پس از اتصال به Raspberry Pi، دستور زیر را برای نصب VNC Server اجرا کنید:

```
sudo apt-get install tightvncserver
```

این دستور VNC Server را با نام کاربری root نصب می کند.

۲. **ایجاد یک رمز عبور برای VNC Server**

پس از نصب VNC Server، باید یک رمز عبور برای آن ایجاد کنید. برای این کار، دستور زیر را اجرا کنید:

```
tightvncserver
```

این دستور یک session جدید برای VNC Server ایجاد می کند. در هنگام ایجاد session، از شما خواسته می شود که یک رمز عبور برای VNC Server وارد کنید.

رمز عبوری را انتخاب کنید که به یاد ماندنی باشد و امنیت بالایی داشته باشد.

۳. **نصب VNC Client روی کامپیوتر دیگر**

برای اتصال به VNC Server از طریق شبکه، باید VNC Client را روی کامپیوتر دیگر نصب کنید.

VNC Client نرم افزاری است که به شما امکان می دهد از طریق شبکه به یک سرور VNC متصل شوید.

VNC Client های زیادی وجود دارد که می توانید از آنها استفاده کنید. یکی از VNC Client های محبوب، TightVNC Viewer است.

برای نصب TightVNC Viewer، می توانید از وب سایت TightVNC دانلود کنید.

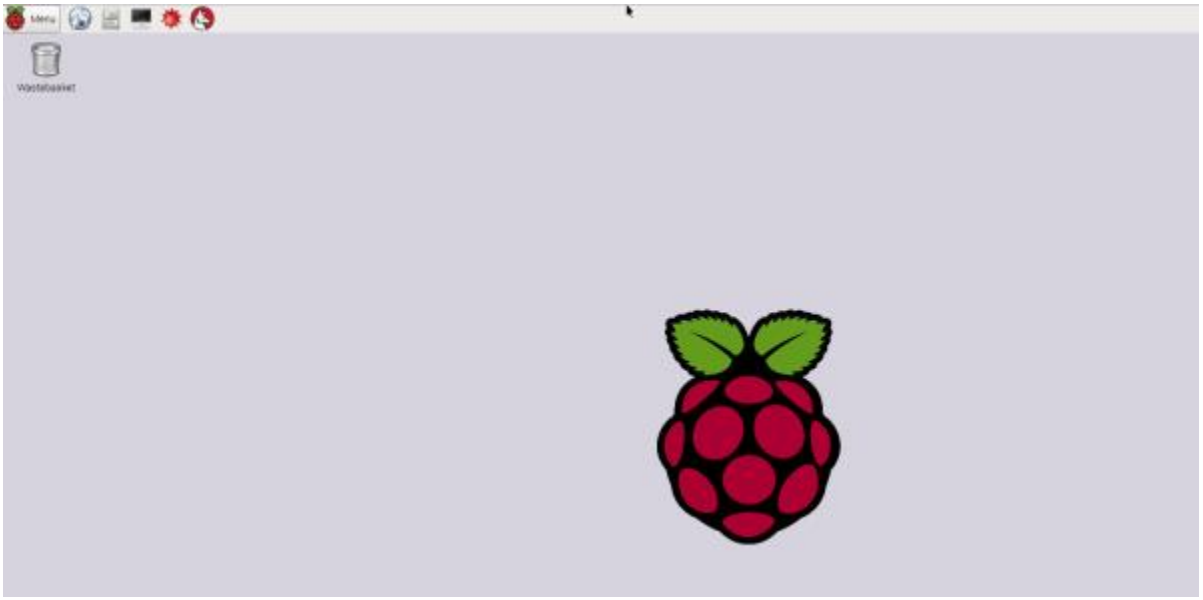
۴. اتصال به VNC Server

پس از نصب VNC Client روی کامپیوتر دیگر، می توانید به VNC Server متصل شوید. برای این کار، VNC Client را اجرا کرده و آدرس IP Raspberry Pi را وارد کنید. همچنین باید رمز عبوری را که در مرحله ۲ ایجاد کردید، وارد کنید.

پس از وارد کردن آدرس IP و رمز عبور، VNC Client به VNC Server متصل می شود. اکنون می توانید از طریق شبکه به رابط کاربری گرافیکی Raspberry Pi دسترسی داشته باشید.

نتیجه گیری

با طی مراحل بالا، می توانید VNC Server را روی Raspberry Pi راه اندازی کنید. VNC Server به شما امکان می دهد رابط کاربری گرافیکی Raspberry Pi را از راه دور کنترل کنید. این می تواند برای کاربردهایی مانند کنترل یک دوربین امنیتی از راه دور یا دسترسی به یک رایانه از راه دور مفید باشد.



آشنایی با زبان C و Assembly Part2.

در این بخش می بایست عملیات ضرب دو عدد ۵ و ۸ را با ۳ روش به زبان اسمبلی ARM پیاده سازی کنیم.
همچنین با تعدادی از دستورات shell آشنایی داده شد. که از آنها در ذیل استفاده شده است.

۱. **اسمبلی چیست ؟**

زبان اسمبلی یک زبان برنامه نویسی سطح پایین است که مستقیماً با سخت افزار کامپیوتر ارتباط برقرار می کند. در زبان اسمبلی، هر دستورالعمل یک کد عددی است که مستقیماً توسط واحد پردازش مرکزی (CPU)پردازش می شود.
زبان اسمبلی برای برنامه نویسی کارهایی که به عملکرد مستقیم سخت افزار نیاز دارند، مانند درایور دستگاه یا سیستم عامل، مفید است. همچنین می تواند برای بهینه سازی کد نوشته شده در زبان های برنامه نویسی سطح بالاتر، مانند C یا C++, استفاده شود.

۲. دستورالعمل ها در اسمبلی

در زبان اسمبلی، هر دستورالعمل یک کد عددی است که مستقیماً توسط واحد پردازش مرکزی (CPU)پردازش می شود. این کدها به عنوان "کد ماشین" شناخته می شوند.

۳. زبان اسمبلی دستوراتی برای انجام کارهای زیر دارد:

- انتقال داده بین رجیسترهای CPU
- انجام عملیات ریاضی و منطقی
- کنترل جریان برنامه
- دسترسی به حافظه
- ورودی و خروجی

در اینجا مثالی از یک دستورالعمل اسمبلی آورده شده است:

```
MOV R0, #1
```

این دستورالعمل مقدار ۱ را در رجیستر R0 ذخیره می کند.
در اینجا مثال دیگری از یک دستورالعمل اسمبلی آورده شده است:

```
ADD R1, R0, R2
```

این دستورالعمل مقدار موجود در رجیستر R0 را با مقدار موجود در رجیستر R2 جمع می کند و نتیجه را در رجیستر R1 ذخیره می کند.
زبان اسمبلی نسبت به زبان های برنامه نویسی سطح بالاتر، مانند C یا C++, انعطاف پذیرتر و کنترل بیشتری بر عملکرد سخت افزار ارائه می دهد. با این حال، یادگیری و استفاده از زبان اسمبلی دشوارتر است.

۴. مزایا و معایب زبان اسمبلی

- مزایای زبان اسمبلی:
  - کنترل مستقیم بر سخت افزار
  - نعطاف پذیری بیشتر
  - عملکرد بهتر
- معایب زبان اسمبلی:
  - یادگیری و استفاده دشوارتر
  - زمان توسعه طولانی تر
  - خطاپذیری بیشتر

۵. پیاده سازی ۳ روش برای انجام محاسبه ضرب ۵ در ۸ به زبان اسمبلی ARM

روش ۱: استفاده از دستور mul

```
mohammad@mohammad-VirtualBox:~$ mkdir CW4
mohammad@mohammad-VirtualBox:~$ cd CW4/
mohammad@mohammad-VirtualBox:~/CW4$ mkdir asm
mohammad@mohammad-VirtualBox:~/CW4$ cd asm
mohammad@mohammad-VirtualBox:~/CW4/asm$ touch mul01.s
mohammad@mohammad-VirtualBox:~/CW4/asm$ touch mul02.s
mohammad@mohammad-VirtualBox:~/CW4/asm$ touch mul03.s
mohammad@mohammad-VirtualBox:~/CW4/asm$ nano mul01.s
mohammad@mohammad-VirtualBox:~/CW4/asm$ cat mul01.s
.text
.global _start

_start:
    mov r0, #5        // Load the value 5 into register r0
    mov r1, #8        // Load the value 8 into register r1
    mul r2, r0, r1    // Multiply r0 and r1, store the result in r2
    // Now, r2 contains the result of 5 * 8 which is 40
    mov r0, #1        // Use 1 for system call number for exit
    ldr r1, =40       // Load the result for exit code
    swi 0             // Make a software interrupt to exit
mohammad@mohammad-VirtualBox:~/CW4/asm$
```

روش دوم: استفاده از حلقه و انجام عمل جمع

```
mohammad@mohammad-VirtualBox:~/CW4/asm$ cat mul02.s
.text
.global _start

_start:
    mov r1, #8      // Set counter to 8 in register r1
    mov r2, #0      // Set sum to 0 in register r2

loop_addition:
    add r2, r2, #5 // Add 5 to sum, store the result in r2
    subs r1, r1, #1// Subtract 1 from the counter r1 and update flags
    bne loop_addition // If the zero flag is not set, branch to loop_addition
    // r2 now contains the result of 5 * 8
    mov r0, #1      // Use 1 for system call number for exit
    mov r1, r2      // Use result as exit code
    swi 0           // Make a software interrupt to exit
mohammad@mohammad-VirtualBox:~/CW4/asm$
```

روش سوم: استفاده از دستور شیفت چپ

```
    swi 0           // Make a software interrupt to exit
mohammad@mohammad-VirtualBox:~/CW4/asm$ cat mul03.s
.text
.global _start

_start:
    mov r0, #5     // Load the value 5 into register r0
    lsl r0, r0, #3// Logical shift left by 3 (equivalent to multiplication by 8)
    // r0 now contains the result which is 5 * 8
    mov r1, r0     // Move the result into r1 for exit
    mov r0, #1     // Use 1 for system call number for exit
    swi 0          // Make a software interrupt to exit
mohammad@mohammad-VirtualBox:~/CW4/asm$
```

۶. Shell اسکریپت چیست ؟

Shell به یک واسط کاربری نرم‌افزار کامپیوتری گفته می‌شود که به کاربران امکان می‌دهد با استفاده از دستورات متنی یا گرافیکی با سیستم عامل ارتباط برقرار کنند. در حالت کلی، شِل‌ها به دو دسته تقسیم می‌شوند:

۱. شل متنی یا همان CLI : در این نوع از شل، کاربر از طریق متن و خط فرمان با کامپیوتر ارتباط برقرار می‌کند. این شل‌ها نسبت به محیط‌های گرافیکی منابع کمتری استفاده می‌کنند و معمولاً از طریق یک پنجره ترمینال یا کنسول در دسترس هستند. مثال‌هایی از شل‌های متنی شامل Bash Bourne Again Shell در سیستم‌عامل‌های مبتنی بر UNIX نظیر لینوکس و macOS، و Command Prompt یا PowerShell در ویندوز است.

۲. شل گرافیکی یا همان GUI در این نوع، کاربر با استفاده از المان‌های گرافیکی نظیر پنجره‌ها، دکمه‌ها، منوها و آیکون‌ها با سیستم عامل تعامل می‌کند. شل گرافیکی به کاربران اجازه می‌دهد بدون دانستن دستورات متنی، کارهایی مانند باز کردن برنامه‌ها، جابجایی فایل‌ها و تنظیم تنظیمات سیستم را انجام دهند. رابط گرافیکی کاربری ویندوز، KDE و GNOME از مثال‌های GUI هستند.

شل‌ها همچنین می‌توانند برای اجرای اسکریپت‌ها و خودکار کردن دستورات پیچیده به کار روند و به کاربران پیشرفته اجازه می‌دهند که کنترل دقیق‌تری بر روی سیستم عامل داشته باشند. قابلیت‌ها و دستورالعمل‌های موجود در شل بستگی به سیستم عامل و شل مورد استفاده دارد.

۷. پیاده سازی برنامه ای به زبان اسمبلی که دو خانه از حافظه را می خواند و آن ها را مقایسه می کند. در صورتی که برابر باشند، مقدار ۱ را بر میگرداند.

```
mohammad@mohammad-VirtualBox:~/CW4/asm$ cat comapre01.s
    .section .data              // هش داده برای تعریف مقادیر ثابت یا متغیرهای جهانی
number1:  .word 0x00000005      // مثال: عدد اول (مقدار دلخواه)
number2:  .word 0x00000003      // مثال: عدد دوم (مقدار دلخواه)

    .section .text              // هش کد که دستورات برنامه را شامل می‌شود
    .global _start              // تعریف یک نقطه ورود برنامه برای لینکر

_start:
    ldr r0,=number1             // بارگذاری می‌کنیم number1 را در r0 آدرس
    ldr r1, [r0]                // بارگذاری می‌کنیم r1 را از حافظه و در number1 مقدار

    ldr r0,=number2             // بارگذاری می‌کنیم number2 را در r0 آدرس
    ldr r2, [r0]                // بارگذاری می‌کنیم r2 را از حافظه و در number2 مقدار

    cmp r1, r2                  // را مقایسه می‌کنیم r2 و r1 مقادیر
    movgt r3, #1                // بارگذاری می‌کنیم r3 بود، مقدار 1 را در r2 بزرگتر از r1 اگر
    movle r3, #0                // بارگذاری می‌کنیم r3 بود، مقدار 0 را در r2 کوچکتر مساوی r1 اگر

    // Output result
    // نتیجه را خروجی دهید یا هر کاری که می‌خواهید انجام دهید syscall اینجا می‌توانید با استفاده از .
    // برای سادگی من از هرگونه خروجی از ترمینال یا فایل خودداری می‌کنم .

end_program:                    // برچسب پایان برنامه برای خروج از برنامه
    mov r0, #0                  // خروج بدون خطا
    mov r7, #1                  // syscall (بر روی بعضی پلتفرم‌ها) برای خروج از برنامه
    svc 0                       // صدا زدن سیستم


mohammad@mohammad-VirtualBox:~/CW4/asm$
```

۸. کد زبان C هر دو برنامه اسمبلی قبل را نوشته و با استفاده از GDB آن ها را disassemble کنید .

در این بخش کامنت گذاری فارسی جهت توضیح کد ها استفاده شده

برنامه دوم به زبان C

```c
#include <stdio.h>

int main() {
    int number1 = 5; // عدد اول (مقدار داده‌شده)
    int number2 = 8; // عدد دوم (مقدار داده‌شده)
    int result;      // متغیری برای نگهداری نتیجه عملیات ضرب

    // عمل ضرب دو عدد
    result = number1 * number2;

    // چاپ نتیجه
    printf("Result of multiplication is: %d\n", result);

    return 0; // خروج از برنامه با وضعیت موفقیت‌آمیز
}
```

کد زبان C برنامه دوم:

```c
#include <stdio.h>

int main() {
    int number1 = 5;    // عدد اول (مقدار دلخواه)
    int number2 = 3;    // عدد دوم (مقدار دلخواه)

    // انجام مقایسه و ارجاع مقدار
    int result = (number1 > number2) ? 1 : 0;

    // چاپ نتیجه
    printf("Result: %d\n", result);

    return 0;           // خروج بدون خطا
}
```

برای انجام عمل reassemble در GDB (GNU Debugger)، ابتدا باید کد C را کامپایل و اجرای برنامه را در حالت دیباگ شروع کنیم، سپس ممکن است بخواهیم دستورات ماشین (machine instructions) را که از کد C اصلی حاصل شده‌اند مشاهده و تغییر دهیم. به شکل زیر عمل می‌کنیم:

۱. مد کد C را با استفاده از کامپایلر gcc با پرچم‌های مناسب برای دیباگ، کامپایل می‌کنیم:

```
gcc -g -o multiplication_program multiplication_program.c
```

این دستور یک فایل اجرایی به اسم multiplication_program را با اطلاعات دیباگ تولید می‌کند.

۲. سپس GDB را روی فایل اجرایی مورد نظر اجرا می‌کنیم:

```
gdb ./multiplication_program
```

۳. درون GDB، می‌توانیم دستور disassemble یا disas را استفاده کنیم تا ماشین کد تولید شده برای تابع main را ببینیم:

(gdb) disassemble main

این دستور دستورات اسمبلی که به وسیله کامپایلر از کد C تولید شده را نشان می‌دهد.

اسمبلی کد اول

```
PS C:\Users\LENOVO\Desktop\1> gcc -g -o multiplication_program 1.c
PS C:\Users\LENOVO\Desktop\1> gdb ./multiplication_program
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from C:\Users\LENOVO\Desktop\1\multiplication_program.exe...done.
(gdb) disassemble main
Dump of assembler code for function main:
   0x00401460 <+0>:     push   %ebp
   0x00401461 <+1>:     mov    %esp,%ebp
   0x00401463 <+3>:     and    $0xfffffff0,%esp
   0x00401466 <+6>:     sub    $0x20,%esp
   0x00401469 <+9>:     call   0x4019e0 <__main>
   0x0040146e <+14>:    movl   $0x5,0x1c(%esp)
   0x00401476 <+22>:    movl   $0x8,0x18(%esp)
   0x0040147e <+30>:    mov    0x1c(%esp),%eax
   0x00401482 <+34>:    imul   0x18(%esp),%eax
   0x00401487 <+39>:    mov    %eax,0x14(%esp)
   0x0040148b <+43>:    mov    0x14(%esp),%eax
   0x0040148f <+47>:    mov    %eax,0x4(%esp)
   0x00401493 <+51>:    movl   $0x405064,(%esp)
   0x0040149a <+58>:    call   0x403a80 <printf>
   0x0040149f <+63>:    mov    $0x0,%eax
   0x004014a4 <+68>:    leave
   0x004014a5 <+69>:    ret
End of assembler dump.
(gdb)
```

اسمبلی کد دوم:

```
PS C:\Users\LENOVO\Desktop\1> gcc -g -o multiplication_program 2.c
PS C:\Users\LENOVO\Desktop\1> gdb ./multiplication_program
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from C:\Users\LENOVO\Desktop\1\multiplication_program.exe...done.
(gdb) disassemble main
Dump of assembler code for function main:
   0x00401460 <+0>:     push   %ebp
   0x00401461 <+1>:     mov    %esp,%ebp
   0x00401463 <+3>:     and    $0xfffffff0,%esp
   0x00401466 <+6>:     sub    $0x20,%esp
   0x00401469 <+9>:     call   0x4019e0 <__main>
   0x0040146e <+14>:    movl   $0x5,0x1c(%esp)
   0x00401476 <+22>:    movl   $0x3,0x18(%esp)
   0x0040147e <+30>:    mov    0x1c(%esp),%eax
   0x00401482 <+34>:    cmp    0x18(%esp),%eax
   0x00401486 <+38>:    setg   %al
   0x00401489 <+41>:    movzbl %al,%eax
   0x0040148c <+44>:    mov    %eax,0x14(%esp)
   0x00401490 <+48>:    mov    0x14(%esp),%eax
   0x00401494 <+52>:    mov    %eax,0x4(%esp)
   0x00401498 <+56>:    movl   $0x405064,(%esp)
   0x0040149f <+63>:    call   0x403a80 <printf>
   0x004014a4 <+68>:    mov    $0x0,%eax
   0x004014a9 <+73>:    leave
   0x004014aa <+74>:    ret
End of assembler dump.
(gdb)
```

همان طور که مشاهده می شود اسمبلی ای که توسط کامپایلر ایجاد شده با اسمبلی نوشته شده متفاوت است. دلیل اصلی تفاوت در پردازنده ها می باشد. که تفاوت های جزئی (در بین پردازنده های ریسک) و یا عمده ای مانند کد کامپایلر شده در پردازنده های اینتل، دارند.

۹. نوشتن و اجرای برنامه استک به زبان C

pi@raspberrypi: ~

GNU nano 2.2.6

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
  int data;
  struct node *next;
} node_t;

struct stack {
  node_t *top;
};

void push(struct stack *stack, int data) {
  node_t *new_node = malloc(sizeof(node_t));
  if (new_node == NULL) {
    printf("Error: Out of memory\n");
    return;
  }

  new_node->data = data;
  new_node->next = stack->top;
  stack->top = new_node;
}

int pop(struct stack *stack) {
  if (stack->top == NULL) {
    printf("Error: Stack is empty\n");
    return -1;
  }

  node_t *temp = stack->top;
  int data = temp->data;
  stack->top = temp->next;
  free(temp);

  return data;
}

int peek(struct stack *stack) {
  if (stack->top == NULL) {
    return -1;
  }

  return stack->top->data;
```

^G Get Help ^O WriteOut
^X Exit ^J Justify

pi@raspberrypi: ~

GNU nano 2.2.6

```c
}

int pop(struct stack *stack) {
  if (stack->top == NULL) {
    printf("Error: Stack is empty\n");
    return -1;
  }

  node_t *temp = stack->top;
  int data = temp->data;
  stack->top = temp->next;
  free(temp);

  return data;
}

int peek(struct stack *stack) {
  if (stack->top == NULL) {
    return -1;
  }

  return stack->top->data;
}

void print_stack(struct stack *stack) {
  node_t *temp = stack->top;
  while (temp != NULL) {
    printf("%d ", temp->data);
    temp = temp->next;
  }
  printf("\n");
}

int main() {
  struct stack stack;
  stack.top = NULL;

  push(&stack, 1);
  push(&stack, 2);
  push(&stack, 3);

  printf("Stack: ");
  print_stack(&stack);

  int popped_element = pop(&stack);
```

تست برنامه استک:

```
pi@raspberrypi: ~
int peek(struct stack *stack) {
  if (stack->top == NULL) {
    return -1;
  }

  return stack->top->data;
}

void print_stack(struct stack *stack) {
  node_t *temp = stack->top;
  while (temp != NULL) {
    printf("%d ", temp->data);
        fflush(stdout);
    temp = temp->next;
  }
  printf("\n");
}

int main() {
  struct stack stack;
  stack.top = NULL;

  push(&stack, 1);
  push(&stack, 2);
  push(&stack, 3);

  printf("Stack: ");
  print_stack(&stack);

  int popped_element = pop(&stack);
  printf("Popped element: %d\n", popped_element);

  printf("Stack: ");
  print_stack(&stack);

  printf("Peek: %d\n", peek(&stack));

  return 0;
}

pi@raspberrypi ~ $ gcc stack.c
pi@raspberrypi ~ $ ./a.out
Stack: 3 2 1
Popped element: 3
Stack: 2 1
Peek: 2
pi@raspberrypi ~ $ nano stack.c
```